KTH Computer Science
and Communication

Proof of Work

A load measure approach to cryptographic mitigation of denial-of-service attacks

FREDRIK LILKAER, CHRISTOPHER TELJSTEDT

Bachelor's Thesis
Supervisor: Douglas Wikström

1 Introduction

The Internet continues to grow and has since a time back fulfilled it's early promise to enable a single source to be connected to several millions of geographically dispersed computers. However, as a consequence it has introduced security flaws of a new magnitude, allowing a single computer to potentially be attacked from millions of sources at once.

Denial of Service continues to plague internet services even if they are efficaciously protected against intrusive security breaches, as evidenced by the recent attack on Spamhaus [2]. A denial of service attack is essentially a targeted effort to prevent a service from servicing legitimate requests by draining the underlying computer resources. Such an attack is executed by having each attacking machine performing only small load of the total work, relying on the cumulative work to overload the target system.

In this paper, we focus on the computational approach to combatting denial of service attacks and to improve service survivability. This concept was originally proposed by Dwork and Naor in their report "Pricing via Processing or Combatting Junk Mail" and then reinvented by Back in *Hashcash - A Denial of Service Counter-Measure.*

The "proof of work" is cryptographic in flavor and the idea is essentially to respond with a problem that is moderately hard to compute but easy to verify. Dwork and Naor originally called this a *pricing function* because of it's economic origin. They introduced this concept as a way to fight e-mail spam by increasing the costs of sending spam, thus making e-mail spam economically unfeasible.

1.1 Background

1.2 Problem definition

Proof of Work has been shown to potentially work as a prevention mechanism to at least mitigate the effects of a DoS attack without making an as assumption about the source.[källa] However, Laurie and Clayton concluded in the paper *"Proof of Work" proves not to work*, that PoW on it's own, is not a feasible solution to fighting spam and denial of service attacks. This is because the classical implementation of Proof of Work does not seperate legitimate users from attackers. Hence, problems from a Proof of Work protected system would not discourage abusers of the system without having an unacceptable effect on legitimate users.

1.3 Problem statement

With the problem defined the question at hand is thus if it is possible to develop a Proof of Work protocol that is independent of client characteristics.

- Is there a viable way to implement a Proof of Work system so that the system's resources are accessable by a diverse variety of devices?
- How should the protocol be optimised for low impact on legitimate client behaviour and high impact on malicious behaviour?
- What advantages and disadvantages does proof of work concept bring in practice and in which applications could it be an improvement to current security?

2 Purpose and method

The purpose of this study is to research ways to improve the classical Proof of Work in such a way that legitimate users are less affected by the Proof of Work than the participants of a DoS attack. Furthermore find a way to dynamically scale the required proof of work when dealing with different hardware.

2.1 Scope and delimitations

Scope:
The coverage of this study .....
The study consists of .....
The study covers the .....
This study is focus on .....

Delimitations:
The study does not cover the .....
The researcher limited this research to .....
This study is limited to .....

2.2 Methodology

3 Theoretical approach

4 System Architecture

Since one requirement of a Proof of Work system is that it minimises differences between a wide variety of devices appearing in the wild we needed to support both desktop and laptops with different operating systems as well as cellphones and tablets. In order to minimise the development effort and maximise maintainability of the code base, a multi platform solution was sought for the client part of the demo application.

A web based solution makes the perfect portable application, however the web is not inherently stateful and quite badly imitates an application server which keeps a constantly open socket connection with connected clients. The advent of websockets turn the whole thing around, with the help of DoS nice fellas we were able to write a truly multi-platform PoW client in html and javascript which maintains the generality and plasticity of a natively written socket based application. The javascript implementation for handling th protocol is quite simple:

```
         $("#msg").append("<br/><b>"+msg+"</b><br/>");
}
function append_result(str){
        $("#result").append(str);
}

$(document).ready(function(){
        if (window["WebSocket"]) {
        conn = new WebSocket("ws://{{$}}/ws");
        conn.onclose = function(evt) {
           log("Connection closed.");
        };
        conn.onmessage = function(evt) {
                // alert("Got response: " + evt.data);
                var response = JSON.parse(evt.data);
                if(response["Opcode"] == 1){
                        // alert("Problems is:" + response.Problems);
                        $('#problems')[0].innerHTML = response.Difficulty.Problems;
                        $('#zeroes')[0].innerHTML = response.Difficulty.Zeroes;

                        var solution = find_xs(response.Problems, response.Difficulty.Zeroes);
                        // $("#result").append("<br/>" + solution + "<br/>");
                            var request = { "Problems": solution,
                                                            "Query": response.Query,
                                                            // "Hash": CryptoJS.SHA256(solution + "" + response["Seed'
                                                            "Opcode": 1};
                            conn.send(JSON.stringify(request));
                    } else {
```

The solution finding part also need to be present:

```
function find_x(difficulty, seed){
    var cmp='';
    for(var i=0;i<difficulty;i++){
        cmp+='0';
    }
    var x=0;

    while (true){
        str = x + "" + seed;
        // $("#result").append("<br/><b>"+x+"<br/>");
        // $("#result").append("<br/><b>Calc is "+CryptoJS.SHA256(str).toString(CryptoJS.enc.Hex)+"<br/>");
        // $("#result").append("<br/><b>Cmp part "+CryptoJS.SHA256(str).toString(CryptoJS.enc.Hex).substr(0,difficulty
        if(CryptoJS.SHA256(str).toString(CryptoJS.enc.Hex).substr(0, difficulty) === cmp){
            // alert("seed: " + seed +  "\nfound " + str +" after " +i);
            return x;

        }
        x++;
    }
}
function find_xs(problems, difficulty){
        var res = [];
        for(var i = 0; i < problems.length; i++){
                var temp = {        Seed: problems[i].Seed,
```

To trigger a request to be sent to the server we build the following function which is then registered to the onclick event of a button in the web gui:

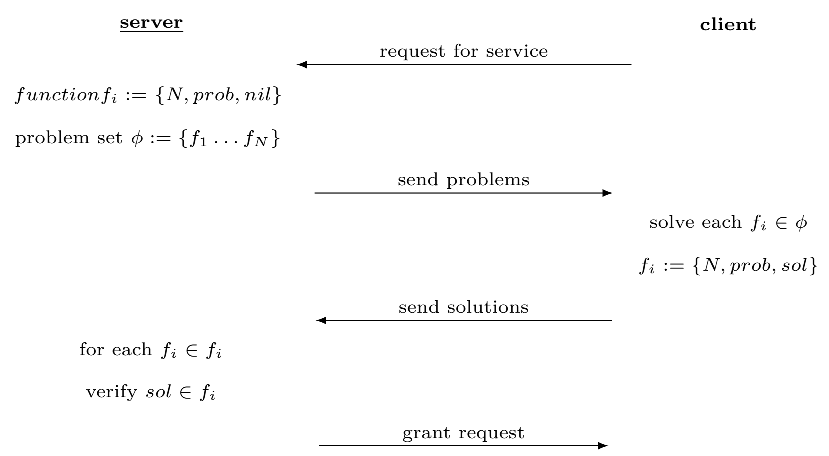
```
}

var startTime;
/*
 * This search function is protected by proof-of-work.
 */
```

For the server side we choose to use Googles novel programming language go(`http://golang.org`). The real strengths of golang in this context is actually not performance nor simplicity[1] but rather it's standard libraries, which in other contexts may appear immature. Golang actually has standard libraries for both http template generation, web serving as well as websockets. This package makes for an ideal platform for an application that needs to deliver the client application[2] to potential clients as well as servicing clients requests in the model application reachable through the websocket interface.

To further ease our programming task, we choose to communicate over the websocket with a single message type which is (de-)serialised (from) to JSON. JSON is natively supported in the Javascript client, and the golang websocket library supports JSON (de-)serialisation which makes for a nice pairing where we have to write absolutely no byte parsing for our protocol.

```
type message struct {
        Opcode, SocketId    int
        Result, Query, Hash string
        Problems            []problem.Problem
        Difficulty          problem.Difficulty
}
```



[1] But the expressiveness, clarity and performance of go programs is not to be dismissed.
[2] HTML, CSS, Javascript and all that magic that make stuff happen in the browser

5 Simulation Experiments

5.1 Attack Model

In this section, we describe the attack model of our simulation experiment. We assume a server *Server*, a network with clients $\{C_i\}$, and attackers *Attacker*.

Assumption 1 *Attacker* cannot ...

Detailed information about assumption 1.

Assumption 2 *Attacker* cannot ...

Detailed information about assumption 2.

6 Results

6.1 Mitigation against Package Dropping

6.2 Mitigation against Server Flooding

6.3 Mitigation against Server Draining

7 Conclusions

7.1 Lessons learned

7.2 Suggested Directions for Future Research

Bibliography

[1] Adam Back. *Hashcash - A Denial of Service Counter-Measure*. Tech. rep. 2002.

[2] BBC News Dave Lee Technology reporter. *Global internet slows after 'biggest attack in history'*. 2013. URL: `http://www.bbc.co.uk/news/technology-21954636` (visited on 04/06/2013).

[3] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *Advances in Cryptology - CRYPTO 92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 139–147. ISBN: 3-540-57340-2. DOI: `http://link.springer.de/link/service/series/0558/bibs/0740/07400139.htm`.

[4] Google. *The Go Programming Language*. 2013. URL: `http://golang.org` (visited on 04/06/2013).

[5] Ben Laurie and Richard Clayton. *"Proof of Work" proves not to work*. 2004.